1. **What is CICD?**

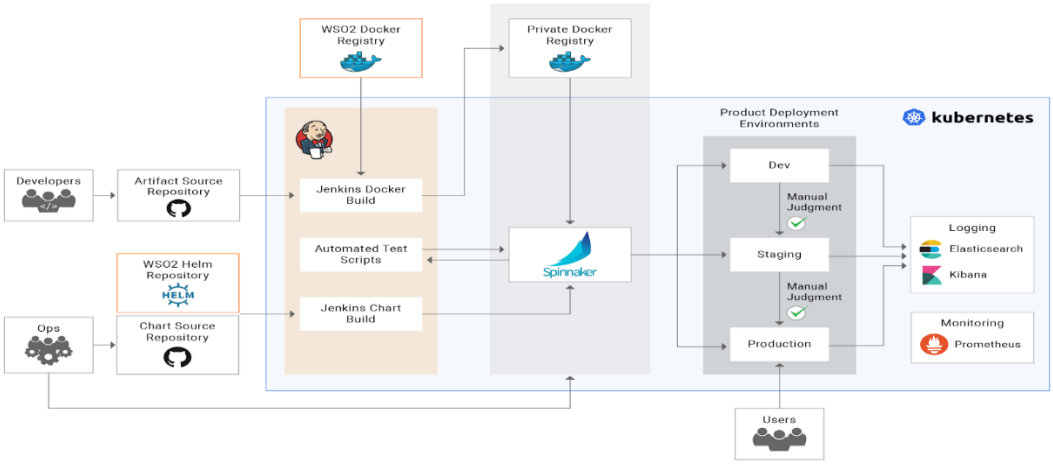
   Continuous Integration and continuous Delivery (CI/CD) is a set of software practices and techniques that enable the frequent release of small batches of code changes, with extensive visibility and traceability. It typically involves the creation of a largely automated pipeline that orchestrates the build, test and deployment of software across staged environments, ultimately leading to deployment in production.

2. **What are all the CICD tools you are using?**

   - ✓ GitHub
   - ✓ Jenkins
   - ✓ Soap
   - ✓ Junit
   - ✓ Nexus
   - ✓ Kibana
   - ✓ App Dynamics
   - ✓ PCF (Pivotal Cloud Foundry)
   - ✓ Gradle

3. **Difference between Continuous integration, Continuous Delivery and Continuous Deployment?**

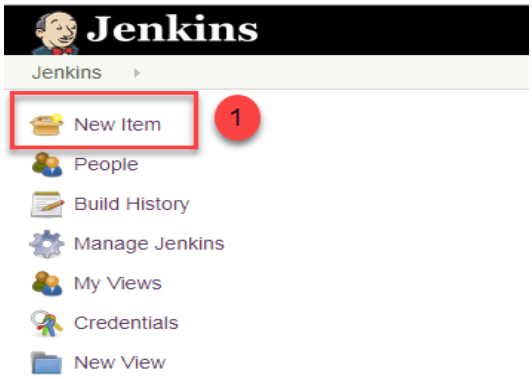
4. **Explain how CICD works internally?**



5. **How to create the Jenkins jobs?**

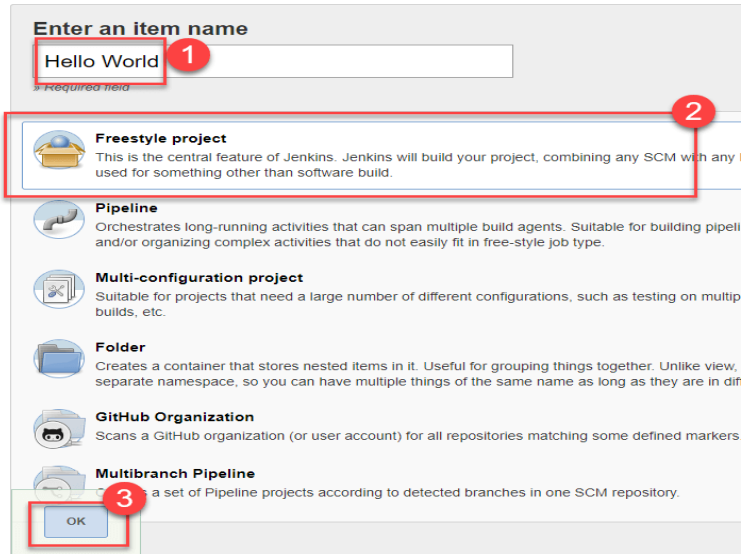
✓ Login to Jenkins

✓ Click on new Item

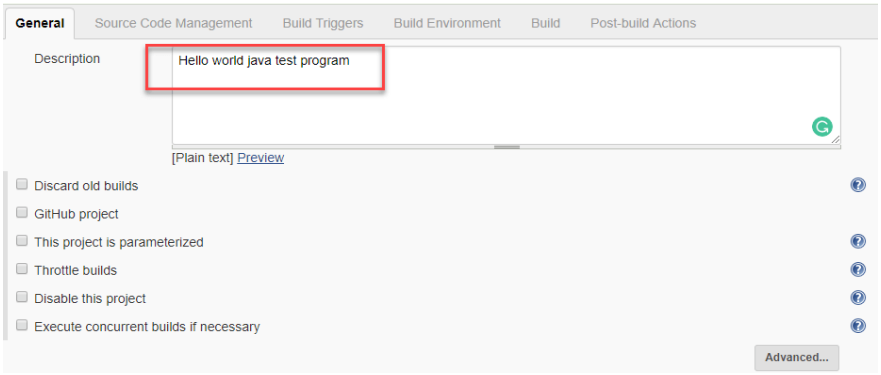


✓ Enter the Item Details

- Enter the name of the item you want to create. We shall use the "Hello world" for this demo.
- Select Freestyle project
- Click Okay

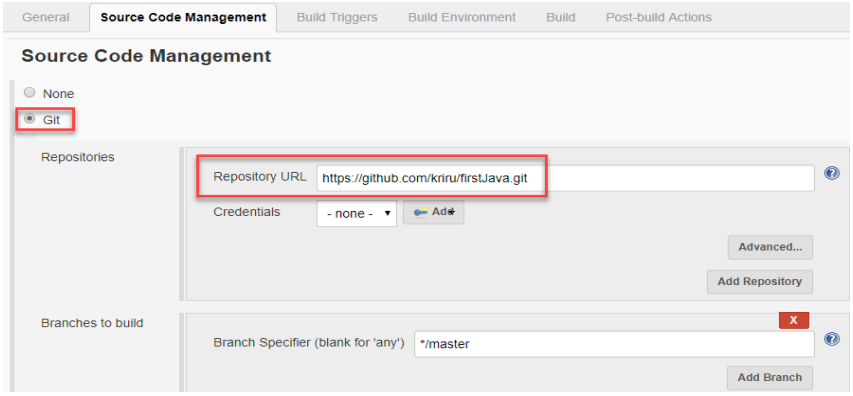


✓ Enter Project Details



✓ Enter repository URL (Git Hub)

If your GitHub repository is private, Jenkins will first validate your login credentials with GitHub and only then pull the source code from your GitHub repository.

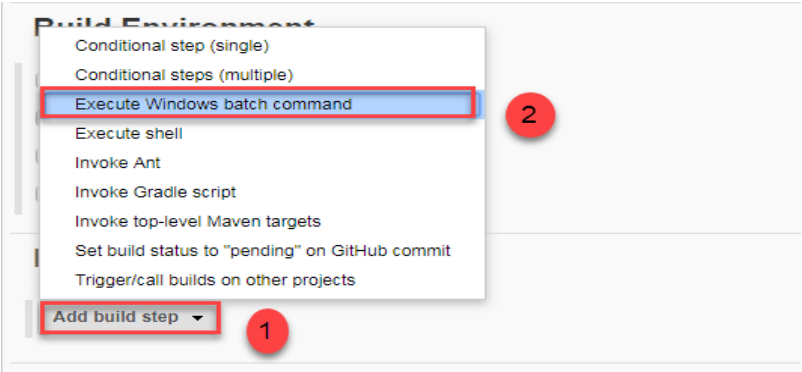


✓ Tweak the settings

Now that you have provided all the details, it's time to build the code. Tweak the settings under the build section to build the code at the time you want. You can even schedule the build to happen periodically, at set times.

Under build,

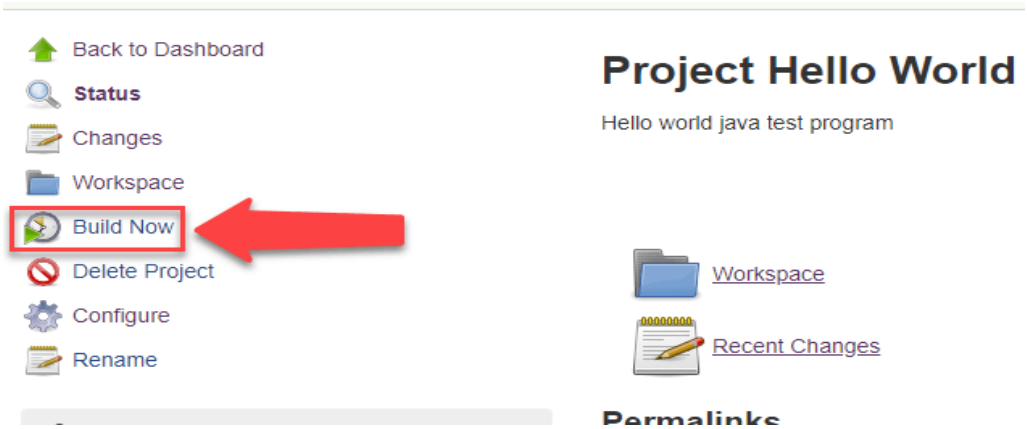
- Click on "Add build step"
- Click on "Execute Windows batch command" and add the commands you want to execute during the build process.



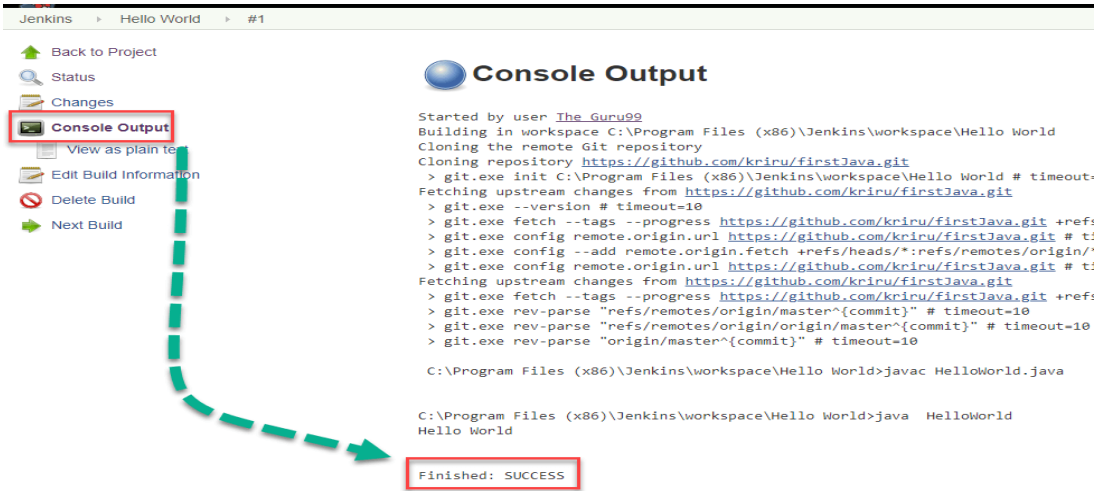
- ✓ Here, I have added the java commands to compile the java code.
- ✓ I have added the following windows commands:

javac HelloWorld.java
java HelloWorld

- ✓ Build Source code

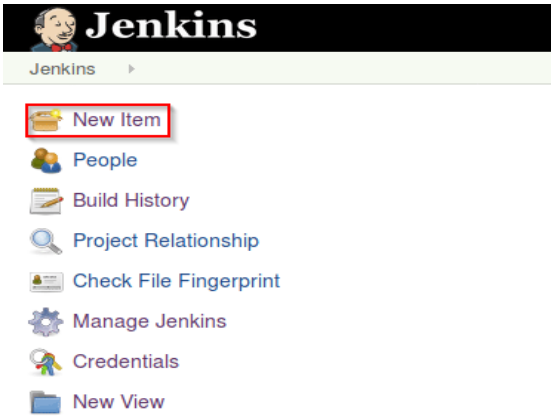


✓ Check the console output for build status.

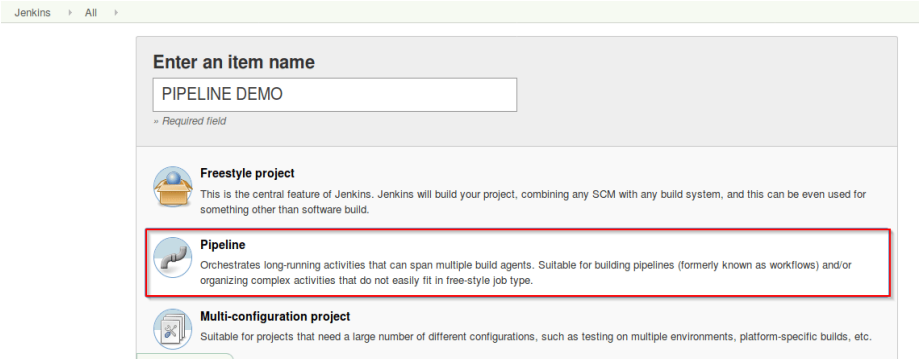


6. How to create the pipeline job using Jenkins?

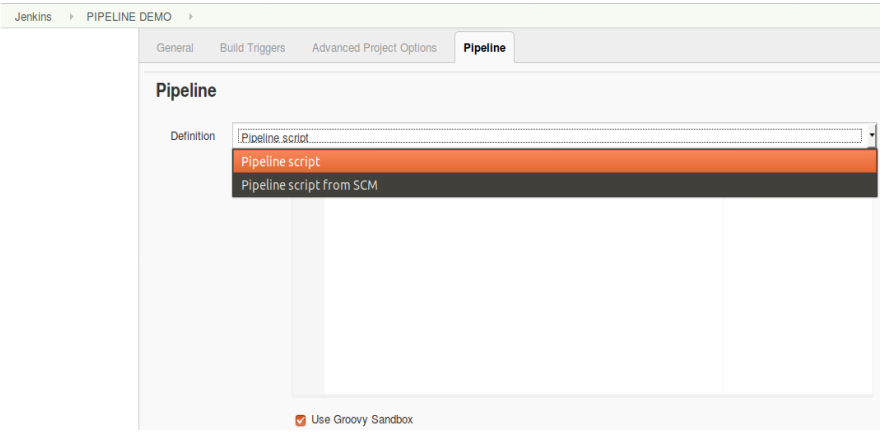
✓ Click on new Item



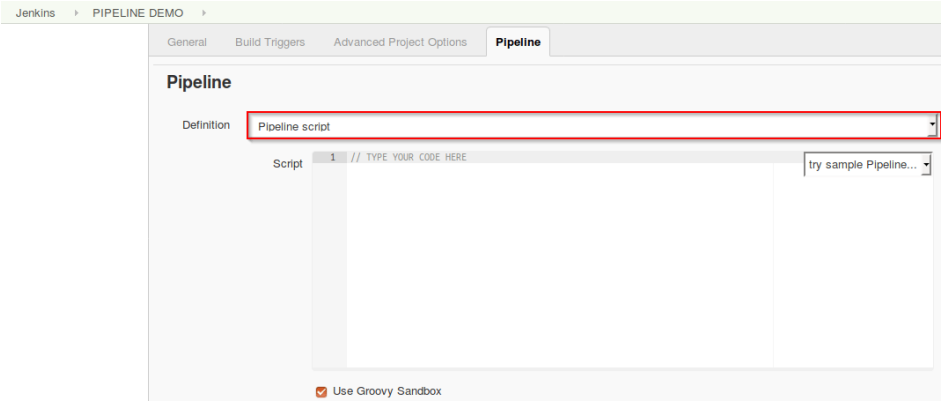
- ✓ Enter a name for your pipeline and select 'pipeline' project. Click on 'ok' to proceed.



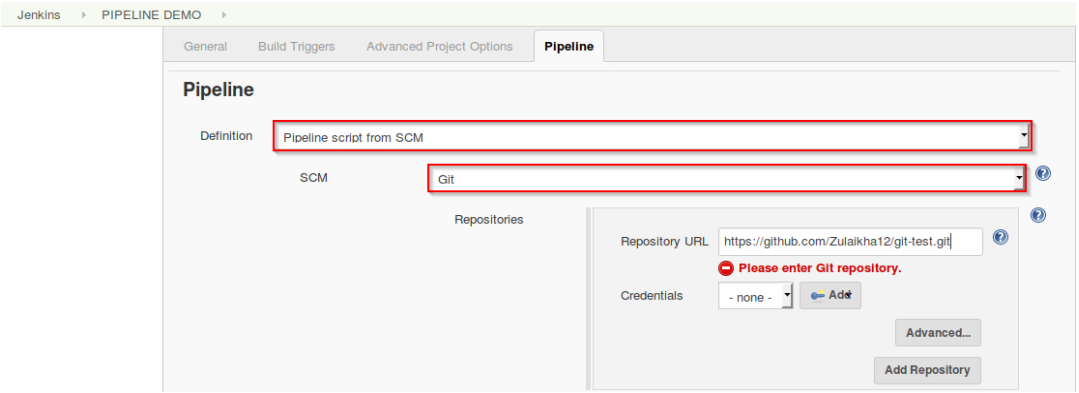
- ✓ Scroll down to the pipeline and choose if you want a declarative pipeline or a scripted one.



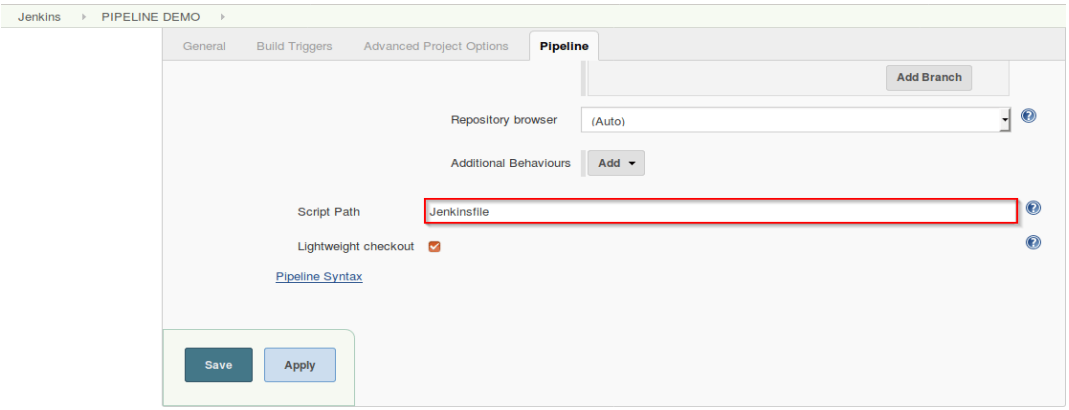
- ✓ If you want a scripted pipeline then choose 'pipeline script' and start typing your code.



- ✓ If you want a declarative pipeline then select 'pipeline script from SCM' and choose your SCM. In my case I'm going to use Git. Enter your repository URL.



- ✓ Within the script path is the name of the Jenkinsfile that is going to be accessed from your SCM to run. Finally click on 'apply' and 'save'. You have successfully created your first Jenkins pipeline.



7. Which kind of Jenkins pipe line you are using for building the release ear?

We are using Cirrus pipe line along with trident compliance for below phases,

- ✓ Check out the code from Git repository
- ✓ Execute the Junits
- ✓ Do the sonar validations code repository
- ✓ Skip the build if Junits/Sonar validations got failed.
- ✓ Do the added jars compliance with Nexus IQ server. If any critical compliance trigger the mail for Job requestor.
- ✓ Validate the Integration test cases
- ✓ Validate the Acceptance test cases
- ✓ For validating the acceptance
- ✓ Mock the corresponding service response with **wire mock** and deploy the web and service mock applications in PCF Zones.
- ✓ After completion of the validations on all api's with wire mock remove the deployed mock apps from PCF.
- ✓ Run the Pen test to identify the security vulranabilities in code.
- ✓ Prepare the release ear and move it to Nexus.
- ✓ In the final step Trident collect all the above evidences and move those as zip folder to nexus and GitHub.

8. Which kind of Jenkins pipe line you are using for deploying the release ear?

We are using Blue-Green deployment for deployment to maintain the zero downtime.

At any time, only one of the environments is live, with the live environment serving all production traffic. For this example, Blue is currently live and Green is idle.

As you prepare a new version of your software, deployment and the final stage of testing takes place in the environment that is not live: in this example, Green. Once you have deployed and fully tested the software in Green, you switch the router so all incoming requests now go to Green instead of Blue. Green is now live, and Blue is idle.

Blue-Green deployment follow below steps,

- ✓ Push an App
- ✓ Update App and Push
- ✓ Map Original Route to Green
- ✓ Unmap Route to Blue
- ✓ Remove Temporary Route to Green

Step1: Push an App:

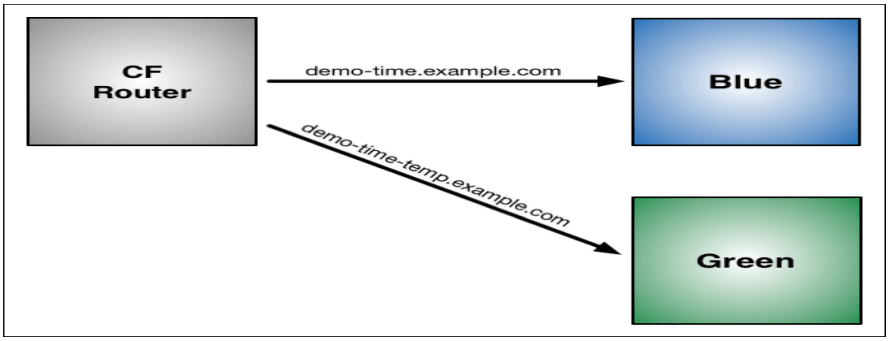
Blue is now running on Cloud Foundry. The CF Router sends all traffic for demo-time.example.com traffic to Blue.

Step2: Update App and Push:

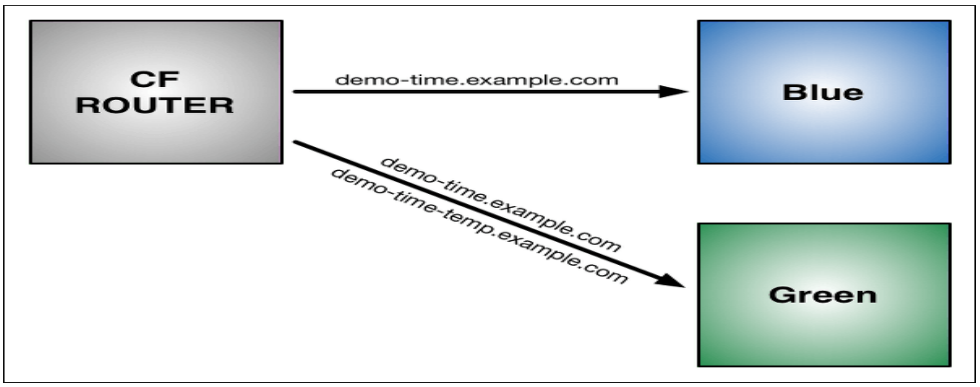
Now make a change to the app. First, replace the word “Blue” on the web page with “Green,” then rebuild the source file for the app. Run cf push again, but use the name “Green” for the app and provide a different subdomain to create a temporary route,

Two instances of our app are now running on Cloud Foundry: the original Blue and the updated Green.

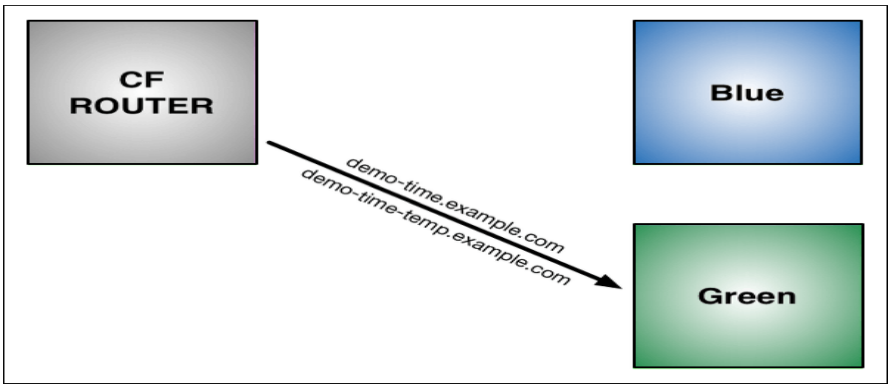
The CF Router continues sending all traffic for demo-time.example.com to **Blue**. The router now also sends any traffic for demo-time-temp.example.com to **Green**.



Step3: Map Original Route to Green



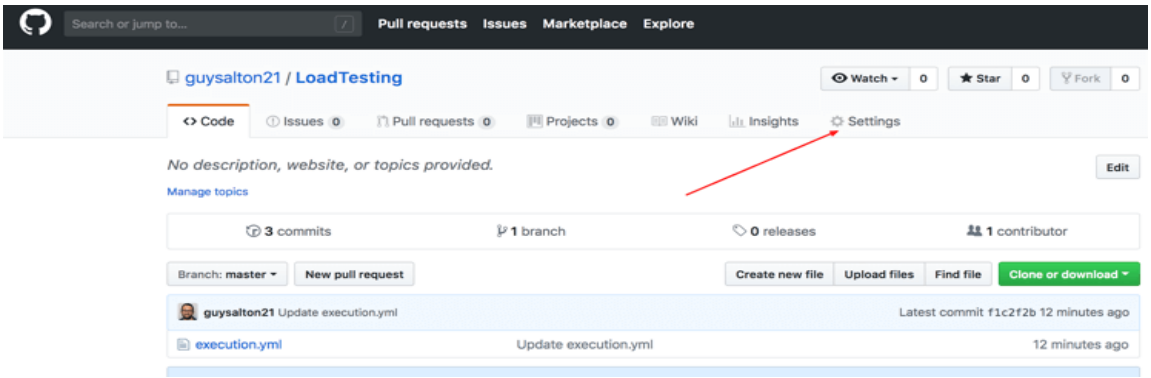
Step4: Unmap Route to Blue



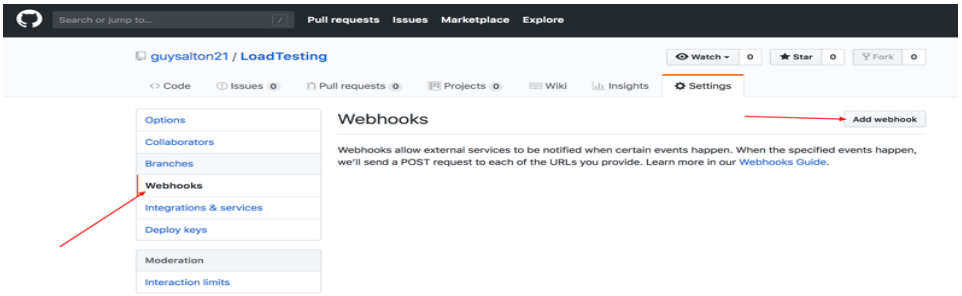
Step5: Remove Temporary Route to Green

9. **Explain the steps to do the once click deployment on code commit from github?**

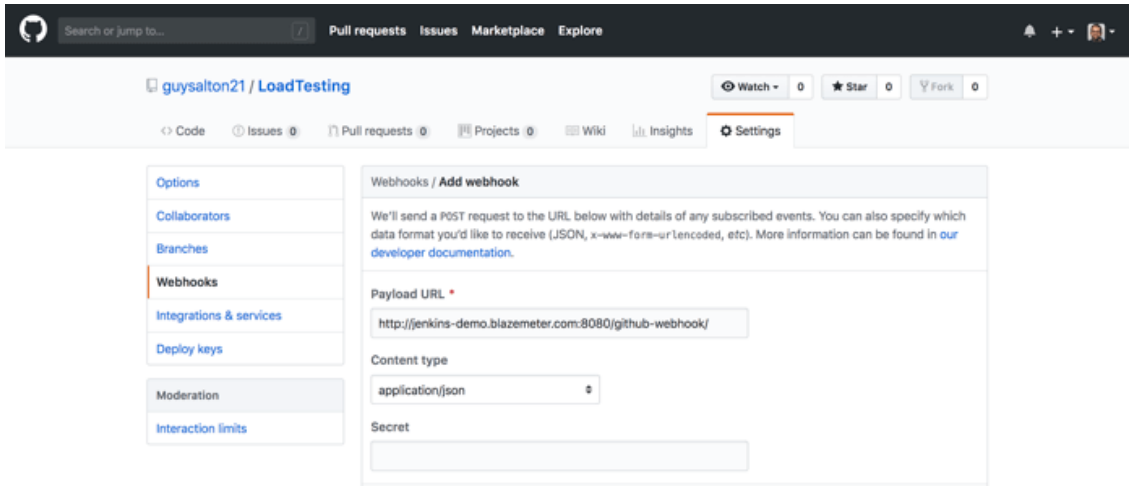
Step1: Go to your GitHub repository and click on 'Settings'



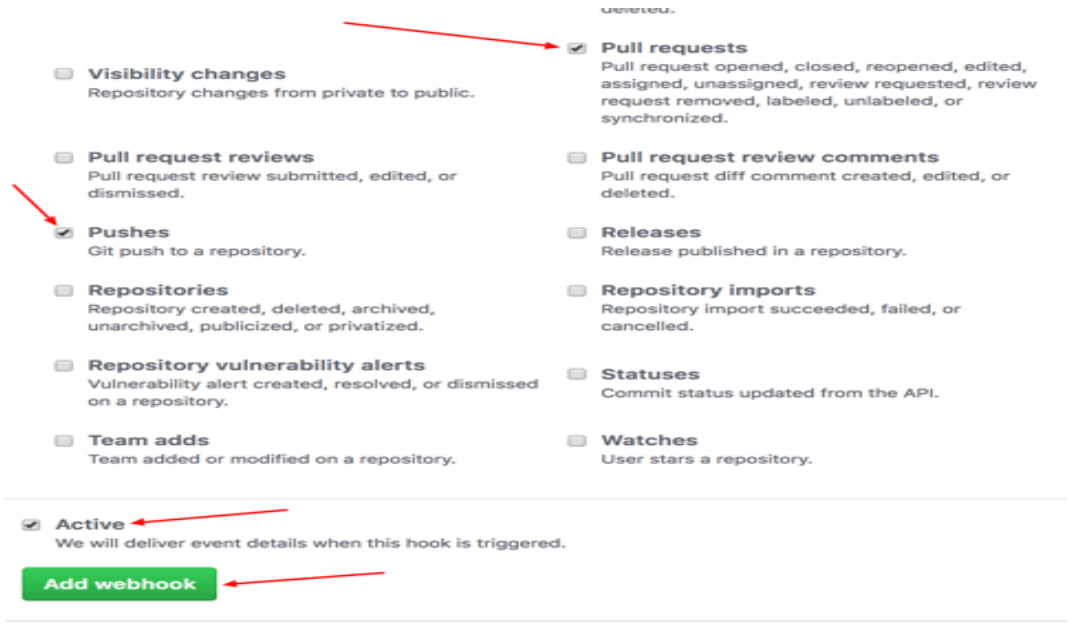
Step2: Click on Webhooks and then click on 'Add webhook'.



Step3: In the 'Payload URL' field, paste your Jenkins environment URL. At the end of this URL add /github-webhook/. In the 'Content type' select: 'application/json' and leave the 'Secret' field empty**.**

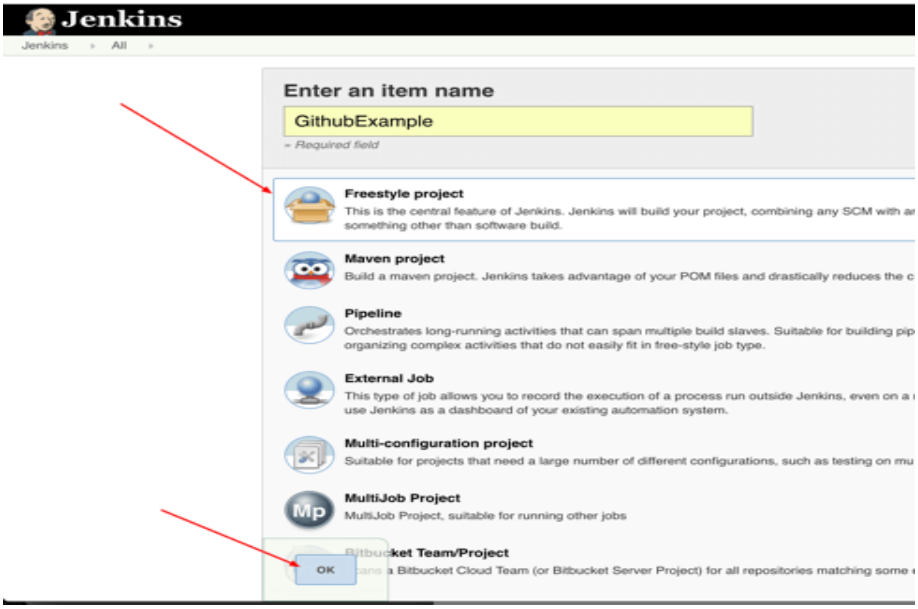


Step4: In the page 'Which events would you like to trigger this webhook?' choose 'Let me select individual events.' Then, check 'Pull Requests' and 'Pushes'. At the end of this option, make sure that the 'Active' option is checked and click on 'Add webhook'.

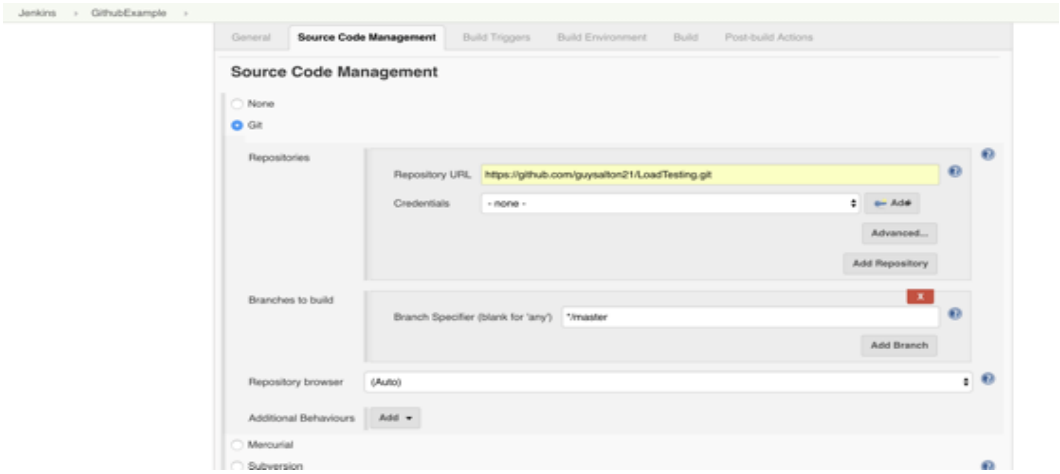


Step6: In Jenkins, click on 'New Item' to create a new project.

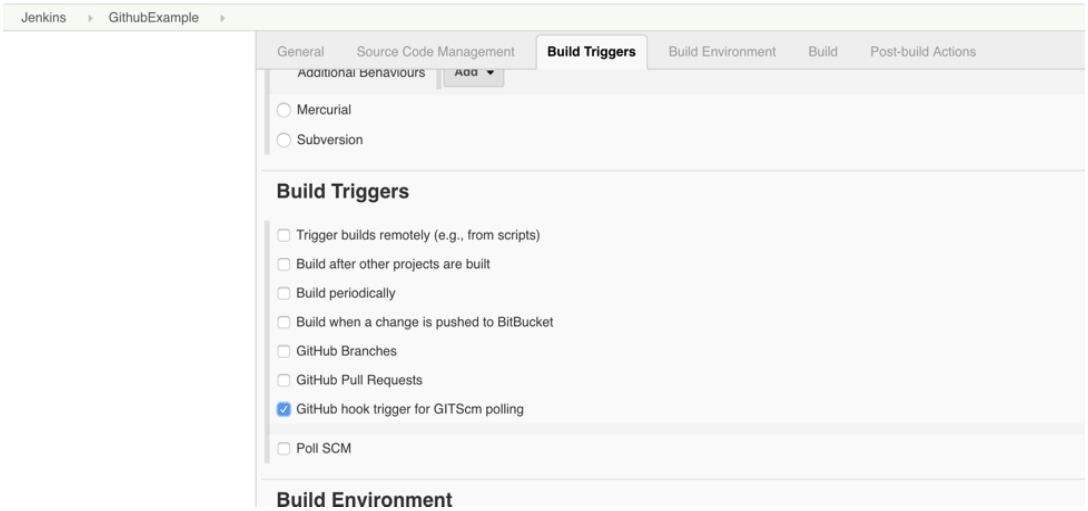
Step 7: Give your project a name, then choose 'Freestyle project' and finally, click on 'OK'.



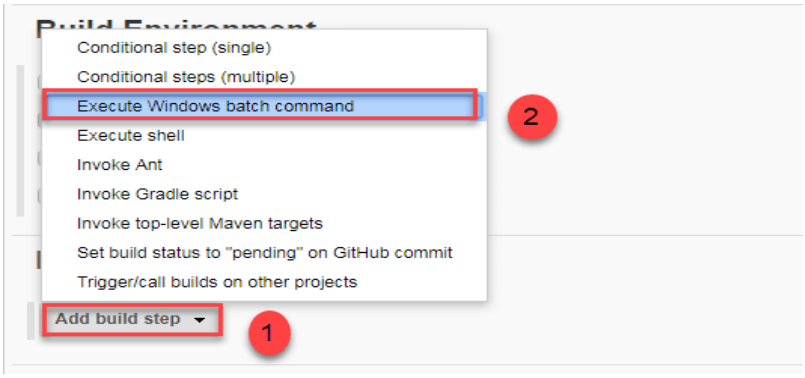
Step8: Click on the 'Source Code Management' tab select git and give the github url in it



Step 9: Click on the 'Build Triggers' tab and then on the 'GitHub hook trigger for GITScm polling'. Or, choose the trigger of your choice.



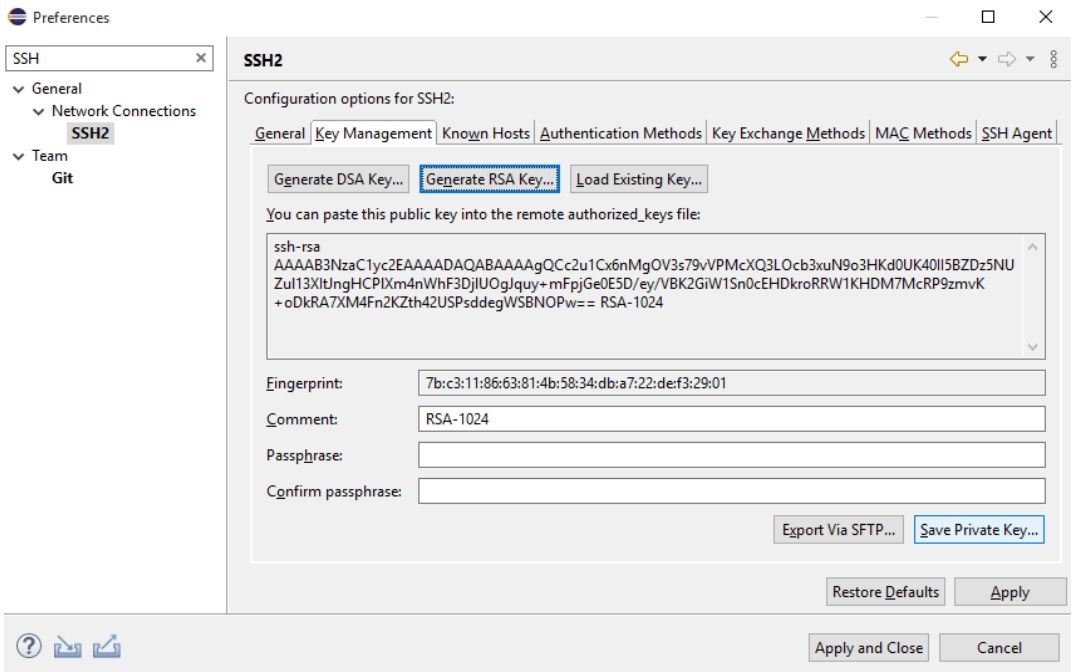
Step 10: Click on the 'Build' tab, then click on 'Add build step' and choose 'Execute windows batch command.



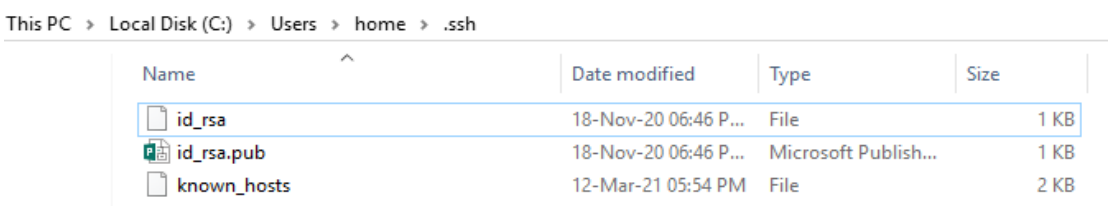
Step 11: In the build give the commands to be executed on your source code

10. How to setup the path to check in the code from local to github?

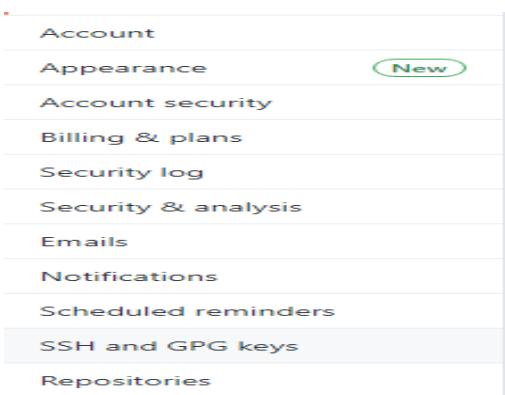
- ✓ In the Eclipse preference tab select the SSH2. Click on Key Management and then click on Generate RSA Key.



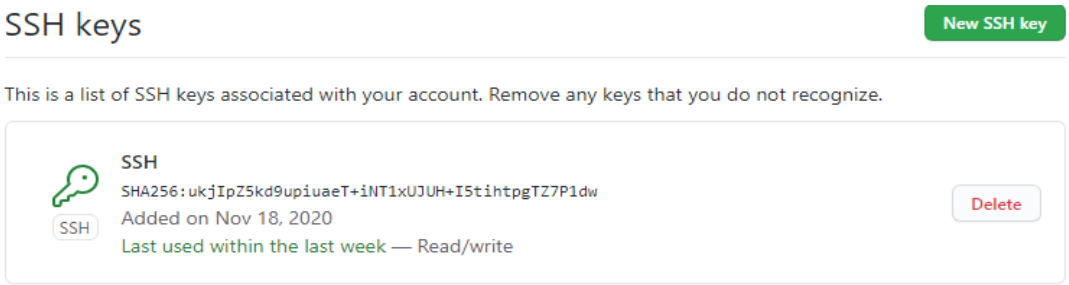
- ✓ Save the Above private key to your system



- ✓ Go to the github settings and select the SSH and GPG keys



✓ Click on new SSH key and paste the key generated from eclipse here then click on save.



11. Explain the limitations on CICD?

Security Vulnerabilities: CI/CD pipelines have a lot of advantages for deploying software builds through an automated process. Unfortunately, a lot of cyber attackers will target CI/CD pipelines because there are usually some security vulnerabilities that are not addressed. A lot of sensitive information can be vulnerable within a CI/CD pipeline, which is why hackers will try to gain access to the pipeline through extreme measures.

Performance Issues: The goal of any CI/CD pipeline will likely be to deliver software and code updates as fast as possible through an automated process. The problem is that a lot of performance issues can make their way into the software if things are not done properly.

Some developers like the idea of implementing an automated testing system to check for possible performance issues. For example, if a batch of code is not performing at an efficient rate, a warning might be pushed back to the developers for additional evaluation. This can help create a defense system against the potential release of a poorly performing software build to customers.

Communication: You will most likely be working with quite a few other people if you are working within a CI/CD pipeline. You might even be broken up into different teams with varying responsibilities. Sometimes the biggest issue within a CI/CD is human communication. For example, if something fails during a software deployment, communication is going to be imperative to solving the issue in a timely manner.

It is plausible to assume that a scenario develops where an automated build test outputs an error and then fails to communicate that information to the appropriate developer, there could be some serious consequences. That is only one reason why communication is so important within this industry.

Version Control: A traditional CI/CD pipeline typically requires a lot of components, processes, and resources to be used. When you finally have your CI/CD pipeline working properly, all of your processes are likely functioning on a stable version. If even a single process gets updated unexpectedly, then your entire CI/CD pipeline deployment process could break.